

# GO-05 01: Асинхронность. Goroutines

## Описание:

В языке Go присутствует довольно простая для использования модель многопоточности. Реализована она с помощью легковесных потоков, называемых горутинами (goroutines). Многопоточность нужна для того, чтобы программа могла выполнять несколько операций одновременно.

Вы уже сталкивались с однопоточными программами, написанными на Go. Например, выведем числа от 0 до 5 (не включительно) в консоль:

```
cmd/myapp/main.go
```

```
package main
```

```
import (
```

```
    "fmt"
```

```
    "time"
```

```
)
```

```
func printNumbers(m, n int) {
```

```
    for i := m; i < n; i++ {
```

```
        fmt.Println(i)
```

```
    }
```

```
}
```

```
func main() {
```

```
    printNumbers(0, 5)
```

```
    fmt.Println("end of print")
```

```
    time.Sleep(time.Second)
```

```
}
```

```
go run main.go
```

Output:

```
0
```

```
1
```

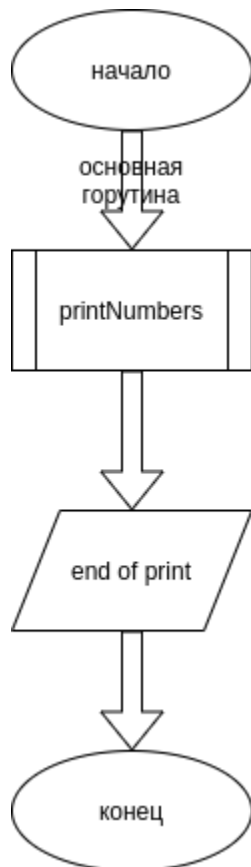
```
2
```

```
3
```

```
4
```

```
end of print
```

Такая программа выполняется последовательно, в один поток, в рамках основной горутины. Сначала вызывается функция `printNumbers`, печатаются соответствующие числа. После этого выводится сообщение "end of print", проходит ожидание одной секунды, после чего программа завершает выполнение. Можно условно изобразить выполнение программы так:



Теперь попробуем запустить вывод чисел и печать сообщения в горутинах. Для этого достаточно добавить `go` перед вызовом функции.

```
package main
```

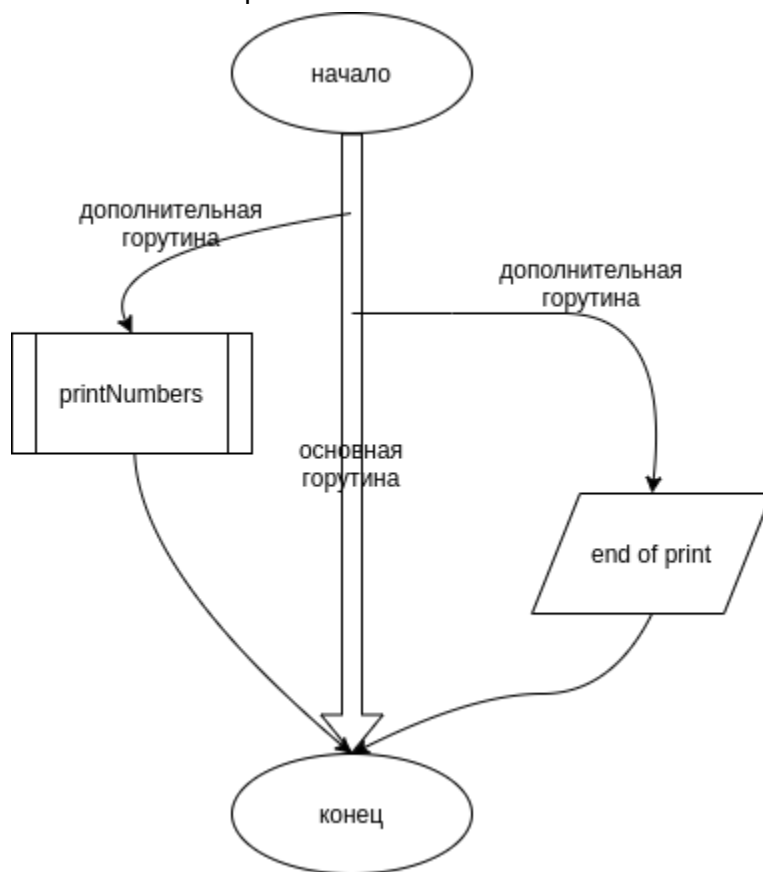
```
import (  
    "fmt"  
    "time"  
)  
  
func printNumbers(m, n int) {  
    for i := m; i < n; i++ {  
        fmt.Println(i)  
    }  
}
```

```
func main() {
    go printNumbers(0, 5)
    go fmt.Println("end of print")
    time.Sleep(time.Second)
}
```

```
go run main.go
```

```
0
end of print
1
2
3
4
```

Обратите внимание, что надпись end of print может оказаться в разных позициях относительно чисел при разных запусках программы. Это пример простого многопоточного приложения.



Возьмем более сложный пример.

```

package main

import (
    "fmt"
    "time"
)

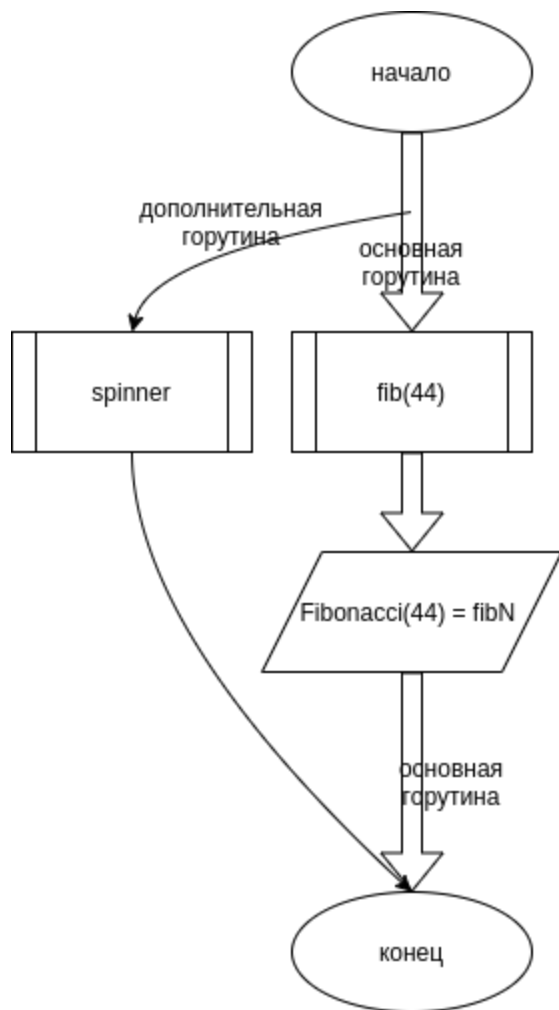
func main() {
    go spinner(100 * time.Millisecond)
    n := 44
    fibN := fib(n)
    fmt.Printf("\rFibonacci(%d) = %d\n", n, fibN)
}

func spinner(delay time.Duration) {
    for {
        for _, r := range `-\|/` {
            fmt.Printf("\r%c", r)
            time.Sleep(delay)
        }
    }
}

func fib(x int) int {
    if x < 2 {
        return x
    }
    return fib(x-1) + fib(x-2)
}

```

В этой программе рекурсивно вычисляется 44ое число Фибоначчи. Это довольно долгая процедура, во время которой с помощью функции `spinner` на экран выводятся (сменяя друг друга) символы `- \| /`, отображающие, что программа продолжает свое выполнение. Если бы мы запустили `spinner` не в горутине, то программа бы выполняла бесконечный цикл, никогда не начав выполнение функции `fib`.



Обратите внимание, что выполнение программы завершается вместе с выполнением основной горутины.

Вызов горутин также доступен для литералов функций:

```
go func() {  
    // do something  
}()
```

## Полезные ссылки:

- [A tour of GO. Concurrency p. 1](#)
- [Understanding Golang and Goroutines](#)
- [Goroutines in GoLang](#)
- [Go by Example: Goroutines](#)

# Задание:

1. Форкните репозиторий с кодом данного задания - [module05](#)
  - в группу `golang_users_repos/<your_gitlab_id>`
2. Создайте у себя в проекте `module05` из ветки `master` ветку `module05_01`.
3. Код для данного задания расположен в файле `fibonachi/main.go` репозитория `module05`
4. Используя функции `spinner` и `fib`, напишите программу, которая параллельно считает 44ое и 45ое числа Фибоначчи, выводя при этом спиннер в консоль, и выводит оба числа в консоль.
5. В ответе пришлите ссылку на МР в ветку `master` своего проекта ветки `module05_01`.